Apple Assembly Line

$1.50

Volume 3 -- Issue 10 July, 1983

In This Issue...

Latest 65C02 Word

The 65C02 really does exist, and we now have a couple of them. As reported inside, the chips we received work in an Apple //e, but not in a][+. Well that seems to be a problem with the NCR chips that I have. Don Lancaster reports that his GTE chips work just fine in all flavors of Apples. I'm swapping an NCR processor for one of his GTE's, and will have more details next month. For the time being, if you buy a 65C02, be sure to get a guarantee that it will work in your Apple.

6502 Mini-Assembler in Applesoft.....Bob Sander-Cederlof

The original Apple II came with a built-in mini-assembler. By typing "F666G" in the monitor, you entered a new realm. The prompt changed from "*" to "!"; errors not only earned a "beep" but also a printed "?"; and monitor commands were still available by typing an initial "$". I learned 6502 programming using this little tool, together with the handy "L" disassembly command. At the time, none of the other computer systems on the market came with either mini-assembler or "L" command.

A mini-assembler allows you to type in mnemonics rather than converting them "by hand". It also will translate branch addresses to the relative offsets needed in relative branch instructions. It will not retain the source code on a file, and will not handle labels. If you want to modify a program, you have to use patches or retype the whole thing. A full assembler will accept labels and comments, and will have some method for working with stored source programs. The S-C Macro Assembler, for example, includes a co-resident source program editor. The extra features a full assembler can include are limited only by the potential market. But mini-assemblers are free.

A long time ago MICRO published a 6502 mini-assembler written in Commodore or OSI BASIC. I started converting it, just for fun, into Applesoft. It wasn't long before I realized that my thought processes were totally incompatible with the author's programming techniques. So I essentially started over. Last month the partially finished listing appeared out of some long forgotten crack, so I dusted it off and finished the program.

It operates a lot like the old "F666G" mini-assembler by Steve Wozniak. (And, even though it is in Applesoft, it is almost as fast.) The initial display is the address "0300" at the left margin, and the cursor in column 20 of the top line on an otherwise empty screen. You can type RETURN to quit, a colon followed by a hex address to change the assembly address, or an instruction mnemonic to be assembled.

I could go into a long-winded explanation of how the program works, describing each subroutine. But you can probably read the listing easily enough, and there are identifying REM statements with each subroutine. The really interesting part to me is the structure of the opcode tables which are contained as strings in OP$, F$, and E$. These tables are set up in lines 2030 through 2050.

OP$ contains the opcodes names. OP$(1) holds the names of all the single byte opcodes. If the input line has no operand data after the opcode mnemonic, the program will search through OP$(1) and had better find your mnemonic. If not, it is "???" for you! Note that the opcode names are three characters each, packed into one long string. Also note that ASL, LSR, ROL, and ROR appear in this string. These four opcodes can have an operand-less mode, as well as any of four modes with operands.

OP$(2) contains the mnemonics for the relative branches.

S-C Macro Assembler (the best there is!).....$80.00
S-C Macro Assembler Version 1.1 Update.....$12.50

S-C Cross Reference Utility.....$20.00
S-C Cross Reference Utility with Complete Source Code.....$50.00

S-C Word Processor.....$50.00
As is, with fully commented source code. Needs S-C Macro Assembler.
Applesoft Source Code on Disk.....$50.00
Very heavily commented. Requires Applesoft and S-C Assembler.
ES-CAPE: Extended S-C Applesoft Program Editor.....$60.00

AAL Quarterly Disks.....each $15.00
Each disk contains all the source code from three issues of "Apple Assembly Line", to save you lots of typing and testing time.

QD#1: Oct-Dec 1980	QD#2: Jan-Mar 1981	QD#3: Apr-Jun 1981
QD#4: Jul-Sep 1981	QD#5: Oct-Dec 1981	QD#6: Jan-Mar 1982
QD#7: Apr-Jun 1982	QD#8: Jul-Sep 1982	QD#9: Oct-Dec 1982
QD#10: Jan-Mar 1983	QD#11: Apr-Jun 1983	

Double Precision Floating Point for Applesoft.....$50.00
Provides 21-digit precision for Applesoft programs.
Includes sample Applesoft subroutines for standard math functions.

FLASH! Integer BASIC Compiler (Laumer Research)..... $79.00
Source Code for FLASH! Runtime Package.....$39.00
Full Screen Editor for S-C Macro Assembler (Laumer Research).....$49.00

The Visible Computer: 6502 (Software Masters).....	(reg. $50.00)	$45.00
Super Disk Copy III (Sensible Software).....	(reg. $30.00)	$27.00
Amper-Magic (Anthro-Digital).....	(reg. $75.00)	$67.50
Amper-Magic Volume 2 (Anthro-Digital).....	(reg. $35.00)	$30.00
Quick-Trace (Anthro-Digital).....	(reg. $50.00)	$45.00
DISASM Dis-Assembler (RAK-Ware).....		$30.00

Blank Diskettes (with hub rings).....package of 20 for $50.00
Small 3-ring binder with 10 vinyl disk pages and disks.....$36.00
Vinyl disk pages, 6"x8.5", hold one disk each.....10 for $6.00
Reload your own NEC Spinwriter Multi-Strike Film cartridges....each $2.50
Diskette Mailing Protectors.....10-99: 40 cents each
100 or more: 25 cents each
ZIF Game Socket Extender.....$20.00
Ashby Shift-Key Mod.....$15.00
Lower-Case Display Encoder ROM.....$25.00
Only Revision level 7 or later Apples.

STB-80 80-column Display Board (STB Systems).....	($249.00)	$225.00
STB-128 128K RAM Card (STB Systems).....	($399.00)	$350.00
Grappler+ Printer Interface (Orange Micro).....	($175.00)	$150.00
Bufferboard 16K Buffer for Grappler (Orange Micro).....	($175.00)	$150.00
Buffered Grappler+ NEW!! Interface and 16K Buffer.....	($239.00)	$200.00

Books, Books, Books.....compare our discount prices!

"THE Book of Apple Software 1983 (with supplement)...	($24.90)	$18.00
"The Apple][Circuit Description", Gayler.....	($22.95)	$21.00
"Enhancing Your Apple II, vol. 1", Lancaster.....	($17.95)	$17.00
"Incredible Secret Money Machine", Lancaster.....	($7.95)	$7.50
"Micro Cookbook, vol. 1", Lancaster.....	($15.95)	$15.00
"Beneath Apple DOS", Worth & Lechner.....	($19.95)	$18.00
"Bag of Tricks", Worth & Lechner, with diskette.....	($39.95)	$36.00
"Apple Graphics & Arcade Game Design", Stanton.....	($19.95)	$18.00
"Assembly Lines: The Book", Roger Wagner.....	($19.95)	$18.00
"What's Where in the Apple", Second Edition.....	($24.95)	$23.00
"What's Where Guide" (updates first edition).....	($9.95)	$9.00
"6502 Assembly Language Programming", Leventhal.....	($18.95)	$18.00
"6502 Subroutines", Leventhal.....	($17.95)	$17.00

Add $1.50 per book for US postage. Foreign orders add postage needed.

*** S-C SOFTWARE, P. O. BOX 280300, Dallas, TX 75228 ***
*** (214) 324-2050 ***
*** We take Master Charge, VISA and American Express ***

OP$(3) holds "JMP" and "JSR". And OP$(4) holds all the rest, which I call the complex opcodes. These are the ones which can have a variety of addressing modes.

F$(1) through F$(4) correspond to OP$(1) through OP$(4). Each three digit group in one of the F$ strings is the opcode value (in decimal) for the corresponding mnemonic from OP$. F$(4) contains a base value, which will be augmented to obtain a specific value for the particular address mode chosen.

The complex opcodes can be classified in many different ways...I tried so many I lost count. I finally settled on the scheme shown in the two tables below:

	+	Imm 08	Zp 04	Abs 0C	Z,X 14	A,X 1C	Z,Y —	A,Y 18	(X) 00	()Y 10	Base	
ADC	0	69	65	6D	75	7D	—	79	61	71	61	097
AND	0	29	25	2D	35	3D	—	39	21	31	21	033
CMP	0	C9	C5	CD	D5	DD	—	D9	C1	D1	C1	193
EOR	0	49	45	4D	55	5D	—	59	41	51	41	065
LDA	0	A9	A5	AD	B5	BD	—	B9	A1	B1	A1	161
ORA	0	09	05	0D	15	1D	—	19	01	11	01	001
SBC	0	E9	E5	ED	F5	FD	—	F9	E1	F1	E1	225
STA	1	—	85	8D	95	9D	—	99	81	91	81	129

	+	Imm 00	Zp 04	Abs 0C	Z,X 14	A,X 1C	Z,Y 14	A,Y 1C	(X) —	()Y --	Base	
ASL	2	—	06	0E	16	1E	—	--	--	--	02	002
LSR	2	—	46	4E	56	5E	—	--	--	--	42	066
ROL	2	—	26	2E	36	3E	—	--	--	--	22	034
ROR	2	—	66	6E	76	7E	—	--	--	--	62	098
BIT	3	—	24	2C	—	--	--	--	--	--	20	032
CPX	4	E0	E4	EC	—	--	--	--	--	--	E0	224
CPY	4	C0	C4	CC	—	--	--	--	--	--	C0	192
DEC	5	—	C6	CE	D6	DE	--	--	--	--	C2	194
INC	5	—	E6	EE	F6	FE	—	--	--	--	E2	226
LDX	6	A2	A6	AE	--	--	B6	BE	--	--	A2	162
LDY	7	A0	A4	AC	B4	BC	—	--	--	--	A0	160
STX	8	—	86	8E	--	--	96	—	--	--	82	130
STY	9	—	84	8C	94	—	--	--	--	--	80	128

The first column of numbers is the opcode class number. These numbers are stored in E$ (see line 2050). The next nine columns show the hex opcode values for each valid combination of opcode and address mode. The last two columns show the "base" value in both hex and decimal.

The top row of numbers (above the dashed lines) shows the augment needed to transform a "base" opcode value into the value for a specific address mode. I broke the data into two separate tables because the Imm and A,Y columns have one pair of values for class 0 and 1 opcodes and another for classes 2 through 9. The class number is used to select which address modes are legal for a given opcode, as well as in selecting the augment values.

If you have ever studied the listing of Wozniak's mini-assembler, you know that his approach was entirely different. If you look inside the S-C Macro Assembler you will find yet another approach. I suppose there are more approaches than existing assemblers. In our line of Cross Assemblers we use about five or six different techniques. The choice depends on the syntax of the operands and the bit structure of the opcodes, as well as whim.

I have also written a disassembler in Applesoft, and the beginnings of a simulator for 6502 code. Maybe they will see print in the near future. There is a lot to be learned from studying or even writing these kinds of programs, and they can even be useful.

```
100  GOTO 2000: REM  MINI-ASSEMBLER FOR APPLE
110  REM ----------------------------------
120  REM PRINT BYTE M AS TWO HEX DIGITS
130 M1 =  INT (M / 16):M2 = M - M1 * 16: HTAB Z:Z = Z + 2: PRINT  MID$ (G
    $,M1 + 1,1); MID$ (G$,M2 + 1,1);: RETURN
140  REM ----------------------------------
150  REM POKE AND PRINT OBJECT BYTE
160 Z = Z + 1: GOSUB 110: POKE AD,M:AD = AD + 1: RETURN
170  REM ----------------------------------
180  REM SET UP TWO-BYTE ADDRESS FIELD
190 HI =  INT (X / 256):LO = X - 256 * HI
200  REM ----------------------------------
210  REM OUTPUT THE OPCODE AND ADDRESS
220 Z = 7: VTAB  PEEK (37):M = OP: GOSUB 160:M = LO: GOSUB 160: IF BY = 3
    THEN M = HI: GOSUB 160
225  GOTO 1000
230  REM ----------------------------------
240  REM OUTPUT SINGLE-BYTE OPCODES
250 Z = 7: VTAB  PEEK (37):M = OP: GOSUB 160: GOTO 1000
260  REM ----------------------------------
270  REM PRINT NEXT ADDRESS
280 Z = 1: CALL  - 868:M =  INT (AD / 256): GOSUB 110:M = AD - M * 256: GOSUB
    110: PRINT ":";: HTAB 20: RETURN
290  REM ----------------------------------
300  REM  CONVERT C$ FROM HEX TO DECIMAL
310 X = 0:I = 0
320 CL$ = "": GOSUB 500: IF CC$ = "#" OR CC$ = "(" THEN CL$ = CC$: GOSUB
    500
325  IF C < 0 THEN CL$ = "ERR": RETURN
330 X = X * 16 + C: GOSUB 500: IF C >  = 0 THEN 330
340 CL$ = CL$ + CC$: GOSUB 500: IF CC$ <  > "" THEN 340
350  RETURN
500  REM ----------------------------------
510  REM GET NEXT CHAR FROM C$
520 I = I + 1: IF I >  LEN (C$) THEN CC$ = "":C =  - 1: RETURN
530 CC$ =  MID$ (C$,I,1):C =  ASC (CC$) - 48: IF (C > 9 AND C < 17) OR C >
    22 THEN C =  - 1: RETURN
540 C = C - 7 * (C > 9): RETURN
900  REM ----------------------------------
910  VTAB  PEEK (37): HTAB 7: PRINT "???" CHR$ (7): REM    ERROR
990  REM ----------------------------------
1000  PRINT : CALL  - 868: GOSUB 260: GOSUB 4000: IF IN$ = "" THEN  STOP

1010  IF  RIGHT$ (IN$,1) = ":" THEN C$ = IN$: GOSUB 290:AD = X: GOTO 1000

1020 L =  LEN (IN$) - 3: IF L < 0 THEN 900
1025  VTAB  PEEK (37): HTAB 6: CALL  - 868: HTAB 20:L$ =  LEFT$ (IN$,3): IF
    L > 0 THEN  GOTO 1050
1030 C$ = "":J = 1: PRINT L$: GOSUB 2100: IF K = 0 THEN 900
1040  GOTO 230
1050  REM ----------------------------------
1051  REM TRY RELATIVE BRANCHES
1052 C$ =  RIGHT$ (IN$,L): PRINT L$" "C$: GOSUB 290: IF CL$ = "ERR" THEN
    900
1053 J = 2: GOSUB 2100: IF K = 0 THEN 1060
1054  IF CL$ <  > "" THEN 900
1056 X = X - AD - 2: IF X <  - 128 OR X > 127 THEN  PRINT "CAN'T BRANCH T
    HAT FAR": GOTO 1000
1057 BY = 2:X = X + 256 * (X < 0): GOTO 190
1060  REM ----------------------------------
```

```
1070  REM TRY JMP AND JSR OPCODES
1080 J = 3: GOSUB 2100: IF K = 0 THEN 1110
1090 BY = 3: IF CL$ = "" THEN 190
1100  IF CL$ = "()" AND OP = 76 THEN OP = 108: GOTO 190
1105  GOTO 900
1110  REM ---------------------------------
1115  REM TRY COMPLEX OPCODES
1120 J = 4: GOSUB 2100: IF K = 0 THEN 900
1130 CA =  VAL ( MID$ (E$,(K + 2) / 3,1)):BY = 2
1140  IF CL$ = "" THEN 1600: REM  ZPORABS
1150  IF CL$ = "#" THEN 1200: REM IMMEDIATE MODE
1160  IF CL$ = ",X" THEN 1400: REM ZP,X OR ABS,X
1170  IF CL$ = ",Y" THEN 1500: REM ZP,Y OR ABS,Y
1180  IF CL$ = "(,X)" THEN 1300: REM (ZP,X)
1190  IF CL$ = "(),Y" THEN 1350: REM (ZP),Y
1195  GOTO 900
1200  REM ---------------------------------
1210  REM IMMEDIATE MODE
1220  IF X > 255 THEN 900
1230  IF CA = 0 THEN OP = OP + 8: GOTO 190
1240  IF CA = 4 OR CA = 6 OR CA = 7 THEN 190
1250  GOTO 900
1300  REM ---------------------------------
1310  REM INDIRECT MODE:  (...,X)
1320  IF CA < 2 THEN 190
1330  GOTO 900
1350  REM ---------------------------------
1360  REM INDIRECT MODE:  (...),Y
1370  IF CA < 2 THEN OP = OP + 16: GOTO 190
1380  GOTO 900
1400  REM ---------------------------------
1410  REM ZP,X OR ABS,X
1420  IF X > 255 THEN 1450: REM ABS,X
1430  IF CA < 3 OR CA = 5 OR CA = 7 OR CA = 9 THEN OP = OP + 20: GOTO 190

1440  GOTO 900
1450  IF CA < 3 OR CA = 5 OR CA = 7 THEN OP = OP + 28:BY = 3: GOTO 190
1460  GOTO 900
1500  REM ---------------------------------
1510  REM ZP,Y OR ABS,Y
1520  IF X > 255 THEN 1540
1530  IF CA = 6 OR CA = 8 THEN OP = OP + 20: GOTO 190
1540 BY = 3: IF CA = 6 THEN OP = OP + 28: GOTO 190
1550  IF CA < 2 THEN OP = OP + 24: GOTO 190
1560  GOTO 900
1600  REM ---------------------------------
1610  REM ZP OR ABS
1620 OP = OP + 4: IF X > 255 THEN OP = OP + 8:BY = 3
1630  GOTO 190
2000  REM ---------------------------------
2010  REM INITIALIZATION
2020  TEXT : HOME : DIM OP$(4),F$(4)
2030 OP$(1) = "ASLBRKCLCCLDCLICLVDEXDEYINXINYLSRNOPPHAPHPPLAPLPROLRORRTIR
     TSSECSEDSEITAXTAYTSXTXATXSTYA"
2032 OP$(2) = "BCCBCSBEQBMIBNEBPLBVCBVS"
2034 OP$(3) = "JMPJSR"
2035 OP$(4) = "ADCANDCMPEORLDAORASBCSTAASLLSRROLRORBITCPXCPYDECINCLDXLDYS
     TXSTY"
2040 F$(1) = "0100000242160881842021362322000742340720081040400421060640960562481201701681861381541 52"
2042 F$(2) = "1441762400482080160801 12"
2044 F$(3) = "076032"
2045 F$(4) = "0970331930651610012251290020660340980322241921942261621601 30128"
2050 E$ = "000000012222344556789"
2060 G$ = "0123456789ABCDEF"
2070 AD = 768: GOTO 1000: REM END OF INITIALIZATION, START AT $0300
2100  REM ---------------------------------
2110  REM SEARCH OPCODE TABLE J
2120 K = 0: FOR I = 1 TO  LEN (OP$(J)) STEP 3: IF L$ =  MID$ (OP$(J),I,3)
     THEN K = I:I = 200:OP =  VAL ( MID$ (F$(J),K,3))
2130  NEXT : RETURN
4000  REM ---------------------------------
4010  REM INPUT A LINE
4020 IN$ = "": CALL 64879:I = 511
4030 I = I + 1: IF I > 550 THEN  RETURN
4040 C =  PEEK (I) - 128: IF C = 13 THEN  RETURN
4050  IF C > 32 THEN IN$ = IN$ +  CHR$ (C)
4060  IF C = 3 THEN  STOP
4070  GOTO 4030
```

Assembly Listing Into a Text File.....Bill Morgan

"That's not a bug, that's a feature!" We've all heard (or said?) that before, but this time it really seems to be true. We have just discovered an undocumented feature in Version 1.1 of the S-C Macro Assembler.

I was trying to see if a program would assemble, and wanted the assembly to be as fast as possible. For some reason I didn't want to do the obvious thing and just switch the listing off, and using a .TA wasn't convenient. So, I stuck a .DU (DUmmy) directive at the beginning of my program, and a .ED (End Dummy)at the end, figuring that would eliminate the time spent writing object code to the disk. When I typed ASM the assembler paused a moment for pass one, then started listing the beginning of the program. But, when the assembler got to the .TF directive the listing stopped and the disk drive started spinning. That wasn't supposed to happen!

When the assembly finished, and the drive stopped turning, I CATALOGed the disk to see what had happened. There was a Binary file, with the filename from my .TF directive, but instead of being much smaller that the source file, it was about twice as big. What could be in that file?

It seemed dangerous to just BLOAD a file that shouldn't exist, so I booted up a disk zap program and inspected the disk. That Binary file contained the text of the assembly listing, starting with the .TF line. Also, the file had no load address and length bytes. The first four bytes were "A0 A0 A0 A0", or the ASCII codes for four spaces. If I had tried to BLOAD the file, it would have loaded at $A0A0, which would have immediately clobbered DOS!

I was preparing a note to warn everybody not to use .TF within a .DU - .ED block, when Bob reminded me of how often we WANT an assembly listing on a Text file, to read into the S-C Word Processor and merge into an article. Why didn't I find out what the Word Processor would make of this file? Well, it read the file just fine, but discarded the first four bytes, since it expects load address and length bytes in a Binary file. In most cases that is no problem, since the first line is the .TF <filename> directive, and will be discarded anyway.

Now we have a Binary file containing the text. That's fine with the S-C Word Processor, but what about other programs, that might require Text files? As it happens, the Macro Assembler creates a Target File as a Text file, then updates the Catalog to turn it into a Binary file. All we have to do is patch the assembler to prevent that change in the file type. That is only a 1-byte patch.

So, all it takes to send the assembly listing to a disk file is to begin the program with a .DU and a .TF, and end it with a .ED. If you want a real Text file, you only have to patch one byte in the assembler.

To make a long story short, here's how to create a Text file containing an assembly listing:

1) At the beginning of your program, put the lines:

```
0000        .DU
0001        .TF LISTING
```

   and at the end put:

```
65535       .ED
```

2) If there is already another .TF directive in your program, insert a "**" at the beginning of the line, to make it into a comment.

3) Enter one of the following patches:

```
$1000 Versions: $29DF:0
$D000 Versions: $C083 C083 EAF9:0 N C080
```

4) Type ASM.

5) And restore the patched location:

```
$1000 Versions: $29DF:4
$D000 Versions: $C083 C083 EAF9:4 N C080
```

Now you can load LISTING into a word processor, delete the first and last lines, and do whatever you want with it!

I tried creating an EXEC file to do all those steps automatically, but ran into trouble. When the assembly ends the EXEC file loses control, and the Text file LISTING doesn't get closed. When I can solve that one I'll let you know.

Osborne Raises Book Prices

Osborne/McGraw-Hill has raised the prices of most of their books. In particular, all of their books which we carry have new higher prices. They are still bargains when you consider how good they are, and how packed with information:

Title	Was	Is Now	Our Price
6502 Assembly Lang Prog	$16.99	$18.95	$18.00
6502 Subroutines	$15.95	$17.95	$17.00
Z-80 Assembly Lang Prog	$16.99	$18.95	$18.00
Z-80 Subroutines	$15.95	$17.95	$17.00

Speeding-up Text File I/O.....Paul Schlyter

In the April 1983 AAL (pages 2-8), Bob Sander-Cederlof presented a small patch that I had sent him almost a year earlier. The patch greatly speeded up LOAD/BLOAD of long files. At the moment, I had recreated a lot of very long assembler source files, such as the source code to DOS and Applesoft. The long assembly times grew annoying, especially when I realized how much time was wasted inside RWTS just waiting for the right sector to pass under the R/W head of the disk drive!

Just one note about what was written on the bottom of page 2 of that issue: my patch does not influence SAVE/BSAVE at all. The read-after-write during a SAVE/BSAVE is made using the VERIFY command, and that command already works at top speed; in fact, VERIFY's speed was a major inspiration for my LOAD/BLOAD patch.

Next I tried to speed up SAVE/BSAVE with an equally simple patch. I found it was not so easy, mainly because SAVE/BSAVE might have to allocate new sectors for the file. I also felt it wasn't worth the trouble writing a more complicated patch, since SAVE/BSAVE isn't really used that often.

Next in line was a speedup of text file read and write. Here I found a great "time-hog" in DOS. The innocent-looking routines at $AE68 and $AE7E each require about 800 cycles to execute. All they do is to swap a 45-byte area back and forth between the file buffers and a local workarea inside the file manager. This is of course necessary when you open/close files or switch from file to file. But if you're reading the same text file, the swapping may not be needed. Nevertheless, file manager swaps the buffer in and out for each and every character you read or write! This amounts to 256*(800+800) = roughly 410,000 cycles or 0.4 seconds for each sector you read or write! This is about six seconds for each track! And all it does during those six seconds is needlessly swap the same 45 bytes back and forth!

The principle of my patch is this: When entering or exiting the file manager, first check to see if you're doing something else besides reading/writing. If so, just go on as usual. If you are reading/writing, check to see if the local workarea belongs to the file being read/written. If so, just exit and save 800 clock cycles. If not, check to see if it belongs to another file. If the workarea contains another file's data, put it back into the file buffer where it belongs and then get the workarea for the current file. All occurs this when you enter the file manager.

Upon exit from the file manager, if you're reading/writing, just set a flag to mark that the local workarea is being used, and save the address of the file buffer it came from. This always saves 800 cycles.

Practical tests show that text file reading/writing is done up to about 40% faster with this patch installed. This is slower

than Diversi-DOS, but on the other hand this patch is compatible with S-C assemblers (and almost everything else in sight). Also, this patch works equally well for all file types; it even speeds up the loading of type-R files with RBOOT/RLOAD (from DOS Tool Kit). Diversi-DOS treats T type files in a special way, but does nothing to speed up type-R files. And mine is free!

I put the patch at $300, because there's no free area large enough inside DOS where you can put it...especially if you have already installed the LOAD/BLOAD speedup described by Bob last April. The listing which follows includes code to hook in the patches by overwriting the file manager where it calls the two workarea transfer subroutines.

```
                    1000 *SAVE S.FAST TEXT (SCHLYTER)
                    1010 *-------------------------------
                    1020 *  SPEEDUP OF DOS TEXT FILE READ/WRITE
                    1030 *  BY PAUL SCHLYTER, SWEDEN, MAY 1983.
                    1040 *-------------------------------
  0042-             1050 PNTR    .EQ $42,43   DOS VARIABLE
                    1060 *-------------------------------
  AE6A-             1070 GET.WRKAREA.FROM.BUFFER .EQ $AE6A
  AE7E-             1080 SAVE.WRKAREA.TO.BUFFER  .EQ $AE7E
  AE81-             1090 SAVE.OLD.WRKAREA        .EQ $AE7E+3
  AF08-             1100 POINT.TO.WRKAREA.BUFFER .EQ $AF08
  B5BB-             1110 FM.OPCODE               .EQ $B5BB
                    1120 *-------------------------------
                    1130         .OR $300
  0300- 4C 60 03    1140         JMP INSTALL
                    1150 *-------------------------------
                    1160 *  PATCH EXECUTED UPON ENTRY TO FILE MANAGER:
                    1170 *       1. IF READ/WRITE AND CORRECT WORK AREA, RETURN
                    1180 *       2. IF WRONG WORK AREA, SAVE OLD WORK AREA
                    1190 *       3. LOAD NEW WORK AREA.
                    1200 *-------------------------------
  0303- AD BB B5    1210 PATCH1 LDA FM.OPCODE      IF NOT READ OR WRITE,
  0306- C9 03       1220         CMP #3                 DO AS USUAL
  0308- 90 16       1230         BCC .1            ...LESS THAN READ/WRITE
  030A- C9 05       1240         CMP #5
  030C- B0 12       1250         BCS .1            ...HIGHER THAN READ/WRITE
  030E- 20 08 AF    1260         JSR POINT.TO.WRKAREA.BUFFER
  0311- A5 42       1270         LDA PNTR      CHECK TO SEE IF CORRECT WORKAREA
  0313- CD 5D 03    1280         CMP PNTR.S         ALREAD LOADED
  0316- D0 08       1290         BNE .1             NO
  0318- A5 43       1300         LDA PNTR+1
  031A- CD 5E 03    1310         CMP PNTR.S+1
  031D- D0 01       1320         BNE .1             NO
  031F- 60          1330         RTS                YES, NOTHING ELSE TO DO,
                    1340 *                          SO EXIT NOW AND SAVE 800 CYCLES!
                    1350 *
                    1360 *       OPCODE NOT READ OR WRITE, OR WRONG WORK AREA.
  0320- 2C 5F 03    1370 .1      BIT FLG       NEED TO PUT BACK THIS WORK AREA?
  0323- 10 11       1380         BPL .2        NO, JUST GET NEW ONE
  0325- 18          1390         CLC           YES, CLEAR FLAG
  0326- 6E 5F 03    1400         ROR FLG
  0329- AD 5D 03    1410         LDA PNTR.S
  032C- 85 42       1420         STA PNTR
  032E- AD 5E 03    1430         LDA PNTR.S+1
  0331- 85 43       1440         STA PNTR+1
  0333- 20 81 AE    1450         JSR SAVE.OLD.WRKAREA
  0336- 4C 6A AE    1460 .2      JMP GET.WRKAREA.FROM.BUFFER
                    1470 *-------------------------------
                    1480 *  PATCH EXECUTED WHEN FILE MANAGER IS FINISHED:
                    1490 *       1. IF READ/WRITE, SET FLAG AND SAVE PNTR
                    1500 *       2. IF NOT R/W, CLEAR FLAG AND SAVE WORK AREA
                    1510 *-------------------------------
  0339- AD BB B5    1520 PATCH2 LDA FM.OPCODE      R/W?
  033C- C9 03       1530         CMP #3
  033E- 90 16       1540         BCC .1            NO
  0340- C9 05       1550         CMP #5
  0342- B0 12       1560         BCS .1            NO
```

```
0344- 38       1570         SEC                  YES, SET FLAG
0345- 6E 5F 03 1580         ROR FLG
0348- 20 08 AF 1590         JSR POINT.TO.WRKAREA.BUFFER
034B- A5 42    1600         LDA PNTR             AND SAVE POINTER
034D- 8D 5D 03 1610         STA PNTR.S
0350- A5 43    1620         LDA PNTR+1
0352- 8D 5E 03 1630         STA PNTR.S+1
0355- 60       1640         RTS                  SAVE ANOTHER 800 CYCLES
0356- 18       1650 .1      CLC                  CLEAR FLAG
0357- 6E 5F 03 1660         ROR FLG
035A- 4C 7E AE 1670         JMP SAVE.WRKAREA.TO.BUFFER
               1680 *
035D- 00 00    1690 PNTR.S  .HS 0000
035F- 00       1700 FLG     .HS 00
               1710 *--------------------------------
               1720 *  TO INSTALL, PATCH DOS LIKE THIS:
               1730 *       .OR $AB0A
               1740 *       JSR PATCH1
               1750 *
               1760 *       .OR $B38E
               1770 *       JSR PATCH2
               1780 *
               1790 *  HERE IS ONE WAY TO DO IT:
               1800 *--------------------------------
               1810 INSTALL
0360- A9 20    1820         LDA #$20      JSR OPCODE
0362- 8D 0A AB 1830         STA $AB0A
0365- 8D 8E B3 1840         STA $B38E
0368- A9 03    1850         LDA #PATCH1
036A- 8D 0B AB 1860         STA $AB0B
036D- A9 03    1870         LDA /PATCH1
036F- 8D 0C AB 1880         STA $AB0C
0372- A9 39    1890         LDA #PATCH2
0374- 8D 8F B3 1900         STA $B38F
0377- A9 03    1910         LDA /PATCH2
0379- 8D 90 B3 1920         STA $B390
037C- 60       1930         RTS
               1940 *--------------------------------
```

Making Paul's Patches Fit in DOS.....Bob Sander-Cederlof

Don't tell me it won't fit! It is so good, it MUST fit!

Let's see...there are 74 bytes available from $B6B3 thru $B6FC. But Paul's patches are 93 bytes long. Maybe if I twist it sideways and then hold my mouth just right....

Ha! It worked!

Let me tell you how, but please don't think I am trying to pick Paul apart. His analysis and creative programming are terrific! He has taught me a lot.

First I noticed some common code in PATCH1 and PATCH2. I made a subroutine called CHECK.OPCODE to test for the read or write command. I used the carry status to pass back the answer to the caller. Then I put the call to POINT.TO.WORKAREA (which loads an address into $42 and $43) at the top of the subroutine. There's no need to duplicate it in the two callers. These changes saved two or three bytes, for a tiny penalty in speed.

I noticed Paul used CLC, ROR FLAG to clear the sign bit of FLAG. I save one byte two times by replacing these with LSR FLAG. I set up the carry status info in CHECK.OPCODE so that carry SET means read/write...this lets me omit the SEC before ROR FLAG when I want to turn on the sign bit.

I noticed that both patches used the current contents of PNTR: PATCH1 compared PNTR to PNTR.SAVE, while PATCH2 copied PNTR into PNTR.SAVE. So I loaded up the contents of PNTR into the A- and X-registers inside my CHECK.OPCODE subroutine. This saves a few more bytes.

At lines 1320-1330 in Paul's program he uses BNE to jump around an RTS. I changed that to BEQ to an existing RTS further down in the program, saving one byte.

I moved the PNTR.SAVE variable, two bytes, to another area. $B5CF and $B5D0 are unused, at the end of the file manager parameter list. Conveniently, the subroutines which load addresses into PNTR refer to three such addresses inside the parameter list. (See the code at $AF08-$AF1C.) The X-register is loaded with 0, 2, or 4 to index into the list. By putting PNTR.SAVE at the end of the list, I can load the X-register with 8 (PNTR.SAVE-$B5C7) and use the same subroutine, entering at $AF12. This takes five bytes instead of twelve for LDA-STA-LDA-STA.

The final shortener I applied was to make the code which clears FLAG and copies the workarea to a buffer into a subroutine. This is called PATCH4 in my listing. The two lines at PATCH4 look just like what was in line inside the PATCH1 code, but different from what was done by the PATCH2 code.

GOT A FUNNY DISK?

... WANT TO KNOW MORE ABOUT IT?

Then you need the

(Confidential Information Advisors)

CAN YOU ...

- ★ **edit normal or protected disks?**
- ★ **quickly find and recover any intact file, however badly the disk is corrupted?**
- ★ **list programs directly from any disk - protected or not?**
- ★ **examine textfiles directly from any disk - protected or not?**
- ★ **analyse the formatting of normal or protected disks?**
- ★ **decrypt commercial software - or encrypt your own?**
- ★ **rapidly auto-search normal or protected disks for anything you like?**
- ★ **understand & use the latest copy protection methods?**
- ★ **use your Apple as a powerful document retrieval system?**
- ★ **make use of an exhaustive knowledge of disk lore?**

YOU CAN NOW — with a little help from these 5 sophisticated disk utilities:

TRICKY DICK examines, records, deletes, and edits. It can: (1) read individual sectors from normal and most protected disks, (2) list their contents in BASIC, assembler, ASCII, or hex, (3) edit them; (4) write them back to the disk. Tricky Dick cunningly bypasses most protection systems, allowing you to work on disks with nonstandard formatting, half-tracks, and altered DOS marks. It is also a chief executive program that directs the following undercover agents:

THE LINGUIST reads in a trackful of raw data for your scrutiny, translates all the address information, and allows you to inspect the track's formatting. It also translates all 3 types of DOS encoding (6 & 2, 5 & 3, 4 & 4), and works with Tricky Dick to list and examine programs or textfiles on any protected disk. You can use The Linguist to recover valuable files from blown disks, improve your programming skills by studying commercial software, and analyse standard or altered formatting.

THE TRACER rapidly searches normal and most protected disks for up to six strings of your choice simultaneously (specified in ASCII or hex). The Tracer also verifies disk formatting, and sniffs out all hidden catalog or VTOC sectors. When it finds something, it transfers control to Tricky Dick and puts the cursor over the object of your search. A few further keystrokes allow you to make any necessary changes and write the sector back to the disk.

THE CODE BREAKER keeps your programs and textfiles from prying eyes by enabling you to translate them into a "secret code" during disk storage. This utility also deciphers encrypted commercial programs, allowing you to use Tricky Dick to read, list, and edit software never before accessible to any disk utility.

THE TRACKER closely shadows the disk drive arm, carefully recording all its movements and operations. The Tracker's job is to display, on either your screen or printer, a list of every track and sector accessed during a LOAD, RUN, SAVE, or any other DOS operation. This utility also tells you exactly where a read or write occurred during any disk access. Use The Tracker's services to locate the precise trouble spots on a clobbered disk, to determine sector skew patterns, to discover the location of hidden "nibble-count" tracks on protected disks, and to learn much more about how DOS works. You'll be surprised to see just exactly where the disk arm really does go!

What's more, you get permanent access to:

THE CIA FILES, a 50,000+ word book designed to turn you into a disk expert. In addition to complete instructions for the 5 CIA utilities, the book contains an easy-to-follow hand-holding tutorial (written in plain English!) on all aspects of the Apple disk. Using the CIA utilities as your personal guides, you progress step-by-step to total disk mastery. You'll acquire a wealth of skills and information relating to disk repair and file recovery, DOS patches, copy protection, disk formatting, program encryption, and other vital topics. Much of the material has never before appeared in print.

All programs are UNPROTECTED, and hence can be copied, listed, and modified at will. (special patches are described in the manual). They require one drive, DOS 3.3, and 48K of RAM.

TO GET THE CIA ON THE TRAIL OF YOUR DISKS, SEND $65.00 TO:

Send $65.00 Money Order, (Checks allow time to clear), to:
Golden Delicious Software Ltd., 350 Fifth Avenue, New York, NY 10001

Dealers enquiries invited

PATCH2 falls into PATCH4 if the opcode was not read/write. This used to clear the flag and call $AE7E; now it is $AE81. Since the difference between $AE7E and $AE81 is a JSR to setup PNTR with the workarea address, and since that was already arranged by CHECK.OPCODE, I can safely enter at $AE81.

No doubt if you followed me this far, you can see even more ways to save bytes. In fact, I see one extra byte myself! But the program is now just the right size for that hole at $B6B3, so enough is enough.

My listing includes some code to install the patches. If you assemble my version, and BSAVE it on a binary file (A$300,L$6A), you can BRUN it whenever you want to install the patches. Or, with version 1.1 of the Macro Assembler just add these lines:

```
1195        .TF B.FAST TEXT
1380        .PH $B6B3
1790        .EP
```

I also worked out the code for using Applesoft POKEs to patch it all in, and here it is:

```
100  REM  TEXT FILE SPEEDUP PATCH
110  READ N
   : IF N = 0 THEN  END
120  READ A
   : FOR I = 0 TO N - 1
   : READ X
   : POKE A + I,X
   : NEXT
   : GOTO 110
200  DATA 74,46771,32,210,182,144,10,205,207,181,
          208,5,236,208,181,240,51,44,252,182,16,
          8,162,8,32,18,175,32,246,182,76,106,174,
          32,8,175,173
210  DATA 187,181,56,73,3,240,5,73,7,240,1,24,165,
          66,166,67,96,32,210,182,144,10,110,252,182,
          141,207,181,142,208,181,96,78,252,182,76,
          129,174,0
220  DATA 2,43787,179,182
230  DATA 2,45967,231,182
240  DATA 0
```

I tested the patches on a 24-sector text file. The file was created by using the TEXT command in the S-C Macro Assembler. I used EXEC to read it back in. I also wrote a short Applesoft program which read the whole file with GET A$ in a loop. Here are the results:

	NORMAL	PATCHED	CHANGE
TEXT	24 sec	18 sec	25% faster
EXEC	52 sec	34 sec	35% faster
GET A$	30 sec	21 sec	30% faster

```
                        1000 *SAVE S.FAST TEXT (RBS-C)
                        1010 *-------------------------------
                        1020 *       PAUL SCHLYTER'S TEXT FILE SPEED-UP
                        1030 *       AS MODIFIED BY BOB SANDER-CEDERLOF
                        1040 *                        JUNE 8, 1983
                        1050 *-------------------------------
0042-                   1060 PNTR    .EQ $42,43
                        1070 *-------------------------------
AE6A-                   1080 COPY.BUFFER.TO.WORKAREA .EQ $AE6A
AE81-                   1090 SAVE.WORKAREA           .EQ $AE81
AF08-                   1100 POINT.TO.WORKAREA       .EQ $AF08
AF12-                   1110 SETUP.PNTR              .EQ $AF12
B5BB-                   1120 FM.OPCODE               .EQ $B5BB
B5CF-                   1130 PNTR.SAVE               .EQ $B5CF,B5D0
                        1140 *-------------------------------
B6B3-                   1150 PATCH.AREA              .EQ $B6B3
AB0B-                   1160 PATCH.LINK1             .EQ $AB0B
B38F-                   1170 PATCH.LINK2             .EQ $B38F
                        1180 *-------------------------------
                        1190         .OR $300
                        1200 *-------------------------------
                        1210 INSTALL.PATCHES
0300- A2 49             1220         LDX #PATCH.SIZE-1
0302- BD 20 03          1230 .1      LDA PATCH.CODE,X
0305- 9D B3 B6          1240         STA PATCH.AREA,X
0308- CA                1250         DEX
0309- 10 F7             1260         BPL .1
030B- A9 B3             1270         LDA #PATCH1
030D- 8D 0B AB          1280         STA PATCH.LINK1
0310- A9 B6             1290         LDA /PATCH1
0312- 8D 0C AB          1300         STA PATCH.LINK1+1
0315- A9 D2             1310         LDA #PATCH2
0317- 8D 8F B3          1320         STA PATCH.LINK2
031A- A9 B6             1330         LDA /PATCH2
031C- 8D 90 B3          1340         STA PATCH.LINK2+1
031F- 60                1350         RTS
                        1360 *-------------------------------
                        1370 PATCH.CODE
                        1380         .OR $B6B3
                        1390         .TA PATCH.CODE
B6B3- 20 E7 B6          1400 PATCH1  JSR CHECK.OPCODE
B6B6- 90 0A             1410         BCC .1          NOT READ/WRITE
B6B8- CD CF B5          1420         CMP PNTR.SAVE
B6BB- D0 05             1430         BNE .1          NO
B6BD- EC D0 B5          1440         CPX PNTR.SAVE+1
B6C0- F0 1E             1450         BEQ PATCH3      YES, RETURN NOW
B6C2- 2C FC B6          1460 .1      BIT FLAG
B6C5- 10 08             1470         BPL .2
B6C7- A2 08             1480         LDX #PNTR.SAVE-$B5C7
B6C9- 20 12 AF          1490         JSR SETUP.PNTR
B6CC- 20 E1 B6          1500         JSR PATCH4      CLEAR FLAG, SAVE WORKAREA
B6CF- 4C 6A AE          1510 .2      JMP COPY.BUFFER.TO.WORKAREA
                        1520 *-------------------------------
B6D2- 20 E7 B6          1530 PATCH2  JSR CHECK.OPCODE
B6D5- 90 0A             1540         BCC PATCH4      NOT READ OR WRITE
B6D7- 6E FC B6          1550         ROR FLAG        SET SIGN BIT
B6DA- 8D CF B5          1560         STA PNTR.SAVE
B6DD- 8E D0 B5          1570         STX PNTR.SAVE+1
B6E0- 60                1580 PATCH3  RTS
                        1590 *-------------------------------
B6E1- 4E FC B6          1600 PATCH4  LSR FLAG        CLEAR SIGN BIT
B6E4- 4C 81 AE          1610         JMP SAVE.WORKAREA
                        1620 *-------------------------------
                        1630 CHECK.OPCODE
B6E7- 20 08 AF          1640         JSR POINT.TO.WORKAREA
B6EA- AD BB B5          1650         LDA FM.OPCODE
B6ED- 38                1660         SEC
B6EE- 49 03             1670         EOR #3          READ?
B6F0- F0 05             1680         BEQ .1          YES
B6F2- 49 07             1690         EOR #7          WRITE?
B6F4- F0 01             1700         BEQ .1
B6F6- 18                1710         CLC
B6F7- A5 42             1720 .1      LDA PNTR
B6F9- A6 43             1730         LDX PNTR+1
B6FB- 60                1740         RTS
                        1750 *-------------------------------
B6FC- 00                1760 FLAG    .HS 00          MUST START WITH FLAG=0
                        1770 *-------------------------------
004A-                   1780 PATCH.SIZE .EQ *-PATCH1
```

WE HAVE FOUND OUT THE HARD WAY THAT DOS CLOBBERS OUR SAVED POINTER, WITH DISASTEROUS RESULTS. IN THE ASSEMBLY LANGUAGE ADD LDA #20,STA $A1B1 AFTER LINE 1340. TO THE APPLESOFT PROGRAM ON PAGE 15, ADD 235 DATA 1,41393,20

I think you get the most benefit if the un-patched DOS has to work so long between calls to RWTS that the disk motor stops, but the patched DOS keeps the motor alive. You save 0.4 seconds per sector anyway, but you can also save waiting for the motor to come up to speed.

Warning: One danger I noted, and which I am wary of, is that FLAG could get out of sync with reality. For example, if somehow FLAG was set with the sign bit on before ever calling the file manager, it could try to copy the workarea to any-old-place in RAM (or ROM, or I/O space). If you install the patches after booting, there should be no problem. But what happens if you initialize a disk with the patched DOS? I think the flag MIGHT turn out wrong. Maybe a little patch is needed to insure FLAG starts out clear, and is cleared after abnormal exits from file manager (such as RESET).

Last-Minute Warning:
DO NOT DO BRUN B.FAST TEXT FROM AN EXEC FILE. IT'S NOT A GOOD IDEA TO USE A TEXT FILE CHANGE THE WAY TEXT FILES WORK.

65C02 Department.....Bill Morgan

I am holding a brand new NCR65C02A. Now I finally believe that there is such a creature as a 65C02! NCR's version of this processor seems to be the same as GTE's. That is, it has all of the enhancements described in the December '82 issue of AAL, except for the single bit set, reset and branch instructions.

We have tested the chip in the computers here, and there's good news and bad news. As Don Lancaster reported last month, the new chip works perfectly in an Apple //e. You just swap processors and start using new opcodes. However, 65C02 chips do not work in the Apple][or Apple][+.

I am told that the problem lies in the execution of instructions like ASL or INC, which read memory, modify the contents, and write the result back to the same address. The 6502 processor does one read and two writes during such an instruction, which is really incorrect. In the 65C02 this has been changed to the proper combination of two reads and one write.

Unfortunately, the Apple][s rely on the timing of the read-write-write cycle, and the read-read-write cycle is just different enough to cause the system to fail. Hopefully some of the hardware specialists can come up with a modification to the older Apples to allow the use of the enhanced processors.

Let's talk about programming the 65C02. With the new chip in a //e, Bob and I started tearing into the S-C Word Processor. We just went through the code, looking for places to substitute a new instruction for several old ones. Come to find out, the most useful change is the true Indirect addressing mode, in place of Indexed Indirect. That means replacing

```
STY YSAVE
LDY #0
LDA (POINTER),Y
LDY YSAVE          with     LDA (POINTER).
```

That's replacing 8 bytes with 2 bytes. BRA (BRanch Always) and STZ (STore Zero) also came in very handy.

All things considered, Bob has decided to wait for the Rockwell version of the 65C02, because he really wants those single bit set, reset, and branch instructions. At last word Rockwell was expecting to start shipping in August, so it will be at least that long before we have any. NCR's chip costs about $10. The Rockwell chip may cost a little more, if and when. We have noticed ads offering 65C02's for $35, which just goes to show how expensive advertising can be.

Revised Monitor Patch for ASCII Display....Bob Sander-Cederlof

Peter Bartlett gave us a nice patch to the Apple Monitor to add ASCII display to the memory dump command. It was published in the Dec 1981 issue of Apple Assembly Line, pages 18-20. You may remember that Peter's patch over-wrote the cassette tape code. Last summer I received two suggested modifications to Peter's code, and at last I pass them on to you.

Bruce Field, from Rockville, Maryland: "I finally got around to building my own EPROM burner the other day, and one of the first things I did was to modify my F8 ROM to include an ASCII listing with the hex dump. I used the routine originally submitted by Peter Bartlett. I found a minor problem with this code.

"The problem is that I have the modified ROM on an Integer BASIC card, and an unmodified ROM on the mother board. If I am in the modified ROM and want to soft-switch back to the mother board, typing 'C081' should do it. But with Peter's patch location C081 is accessed inside the patch itself, so the card switches off with PC pointing inside the cassette tape code!

"My solution is to leave the loading of the memory location in its original position. This makes the patch slightly longer, but it still fits inside the cassette tape space. Also, since I detest flashing characters, I filter these out. I force control characters to inverse mode, and all others to normal video."

```
                    1000 *SAVE S.MON ASCII DISPLAY (FIELD)
                    1010 *---------------------------------
                    1020 *   PATCHES TO ADD ASCII DUMP TO APPLE MONITOR
                    1030 *   ORIGINAL BY PETER BARTLETT
                    1040 *   MODIFIED BY BRUCE FIELD
                    1050 *---------------------------------
003C-               1060 A1L        .EQ $3C
FDED-               1070 COUT       .EQ $FDED
FDDA-               1080 PRBYTE     .EQ $FDDA
                    1090 *---------------------------------
                    1100         .OR $FDBD
                    1110         .TA $0DBD
FDBD- 20 C9 FC      1120         JSR PATCH    CALL MY PATCH CODE
                    1130 *---------------------------------
                    1140         .OR $FCC9
                    1150         .TA $0CC9
FCC9- 48            1160 PATCH   PHA          SAVE BYTE
FCCA- A5 3C         1170         LDA A1L      LOW BYTE OF DUMP ADDRESS
FCCC- 29 07         1180         AND #7       MASK LINE POSITION
FCCE- 18            1190         CLC
FCCF- 69 1F         1200         ADC #31      COMPUTE HORIZONTAL OFFSET
FCD1- A8            1210         TAY
FCD2- 68            1220         PLA          GET BYTE FROM STACK
FCD3- 48            1230         PHA          KEEP COPY ON STACK
FCD4- 09 80         1240         ORA #$80     FORCE NORMAL VIDEO
FCD6- C9 A0         1250         CMP #$A0     MAKE CONTROL-CHARS INVERSE
FCD8- B0 02         1260         BCS .1       ...NOT CTRL
FCDA- 29 7F         1270         AND #$7F     ...CTRL
FCDC- 91 28         1280 .1      STA ($28),Y  STORE IT ON THE SCREEN
FCDE- A0 00         1290         LDY #0       RESTORE Y
FCE0- 68            1300         PLA          RECOVER BYTE AGAIN
FCE1- 4C DA FD      1310         JMP PRBYTE
```

Brooke Boering, from Schaumburg, Illinois: "Here is a slightly modified version of Peter Bartlett's monitor patch. I modify control characters to display as an underline character, and lower case codes to inverse video. Other characters display in normal video."

```
                        1000 *SAVE S.MON ASCII DISPLAY (BOERING)
                        1010 *-------------------------------
                        1020 *   PATCHES TO ADD ASCII DUMP TO APPLE MONITOR
                        1030 *   ORIGINAL BY PETER BARTLETT
                        1040 *   MODIFIED BY BRUCE FIELD
                        1050 *   MODIFIED AGAIN BY BROOKE BOERING
                        1060 *-------------------------------
003C-                   1070 A1L        .EQ $3C
FDED-                   1080 COUT       .EQ $FDED
FDDA-                   1090 PRBYTE     .EQ $FDDA
                        1100 *-------------------------------
                        1110         .OR $FDBD
                        1120         .TA $0DBD
FDBD- 20 C9 FC          1130         JSR PATCH     CALL MY PATCH CODE
                        1140 *-------------------------------
                        1150         .OR $FCC9
                        1160         .TA $0CC9
FCC9- 48                1170 PATCH   PHA           SAVE BYTE
FCCA- A5 3C             1180         LDA A1L       LOW BYTE OF DUMP ADDRESS
FCCC- 29 07             1190         AND #7        MASK LINE POSITION
FCCE- 18                1200         CLC
FCCF- 69 1F             1210         ADC #31       COMPUTE HORIZONTAL OFFSET
FCD1- A8                1220         TAY
FCD2- 68                1230         PLA           GET BYTE FROM STACK
FCD3- 48                1240         PHA           KEEP COPY ON STACK
FCD4- 09 80             1250         ORA #$80      FORCE NORMAL VIDEO
FCD6- C9 A0             1260         CMP #$A0      MAKE CONTROL-CHARS INVERSE
FCD8- B0 02             1270         BCS .1        ...NOT CTRL
FCDA- A9 DF             1280         LDA #$DF      MAKE CONTROL INTO UNDERLINE
FCDC- C9 E0             1290 .1      CMP #$E0      IN LOWER-CASE RANGE?
FCDE- 90 02             1300         BCC .2        ...NO, DISPLAY NORMAL VIDEO
FCE0- 29 1F             1310         AND #$1F      ...YES, FORCE INVERSE VIDEO
FCE2- 91 28             1320 .2      STA ($28),Y   STORE IT ON THE SCREEN
FCE4- A0 00             1330         LDY #0        RESTORE Y
FCE6- 68                1340         PLA           RECOVER BYTE AGAIN
FCE7- 4C DA FD          1350         JMP PRBYTE
                        1360 *-------------------------------
```

After assembling Bruce's version above, using the S-C Macro Assembler resident in my language card, I installed the patch by typing:

```
$C083 C083      (write enable RAM card)
$FCC9<CC9.CE3M (move the patch into the
                cassette space)
$FDBE:C9 FC     (install patch address in JSR)
```

And it worked! To install Brooke's code I had to move a few more bytes:

```
$FCC9<CC9.CE9M
```

If you have an Apple //e, or a lower case display adapter in an older Apple, you will not want to display lower case characters in inverse mode. Everyone seems to have their own preferences about how to display the 256 possible hex values on Apple's screen. Choose your own favorite!

S-C Macro Assembler

S-C Software Corporation is pleased to introduce the S-C Macro Assembler, the latest version of our most popular product. The S-C Assembler II Version 4.0 already has the reputation of being the easiest editor/assembler to learn, to remember, and to use...now the S-C Macro Assembler provides a new level of power and performance for the beginner and professional programmer alike.

- 29 Commands, including a convenient EDIT command with 15 subcommands. COPY and REPLACE commands further simplify entry and modification of even the most complex programs.

  20 Assembler Directives (Pseudo-Ops) provide all features necessary for professional software development, including conditional assembly and macro generation.

  Operates in any Apple II or Apple II Plus with at least 32K RAM and one disk drive. Any additional memory or disk drives will be used as required. A Language Card version is also included.

  A memory size of 48K allows source programs of over 24,000 bytes to be handled entirely within RAM. The Language Card version allows source programs of over 32,000 bytes. Much larger programs can be edited and assembled using the "INCLUDE" and "TARGET FILE" capabilities, up to the limit of online disk storage.

  Programs can be edited, assembled, and tested entirely within the framework of the S-C Macro Assembler. The editor and assembler are coresident, allowing rapid cycles of modification, reassembly, and check-out. All DOS and Apple Monitor commands are active as well, providing a familiar interface to the standard Apple features.

  Uses its own high-speed technique to store source files, but also can read or write standard TEXT files. You can EXEC in files from another assembler, use some other text editor to prepare files, keep a library of routines on disk to EXEC into any program, or use S-C Macro Assembler to prepare EXEC files for any purpose.

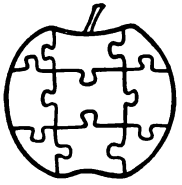
  Price is only $80! Includes diskette with Macro Assembler and sample programs, a 100-page Reference Manual, and a Programmer Reference Card. (Registered Owners of S-C Assembler II Version 4.0 may purchase the upgrade package for only $27.50).

Commands	
Source:	NEW, LOAD, SAVE, TEXT, HIDE, MERGE
Editing:	LIST, FIND, EDIT, DELETE, REPLACE, COPY, RENUMBER
List Control:	FAST, SLOW, PRT,"
Object:	ASM, MGO, VAL, SYMBOLS
Miscellaneous:	AUTO, MANUAL, INCREMENT, MEMORY, MNTR, RST, USR

All Apple Monitor Commands
All Apple DOS Commands

Assembler Directives	
.OR	Origin
.TA	Target Address
.TF	Target File
.IN	Include File
.EN	End of Program
.EQ	Equate
.DA	1- or 2-byte Data
.HS	Hex String
.AS	ASCII String
.AT	ASCII Terminated
.BS	Block Storage
.TI	Title
.LIST	Listing Options
.PG	Page Eject
.DO, .ELSE, .FIN	Conditional Assembly
.MA	Macro Definition
.EM	End of Macro
.US	User Directive

Already well-known for excellent support, S-C Software Corporation pledges to continue development of new features, and to help owners gain the maximum benefit from the S-C Macro Assembler. In addition to telephone consultation for registered owners, a monthly newsletter is available by subscription (currently $15/year). The "Apple Assembly Line" covers items of interest to assembly language programmers at all levels, and has helped many to advance their programming skills.



"Makes assembly language programming on the Apple as easy as programming in BASIC."

S-C Software Corporation
2331 Gus Thomasson, Suite 125
P.O. Box 280300
Dallas, Texas 75228
(214) 324-2050
We take Master Card and Visa
Apple is a trademark of Apple Computer

Answered Prayer.....Bob Sander-Cederlof

Last month's headlines bemoaned our burglary, with equipment worth over $11000 missing from our offices. And un-measurable amounts of software. And a damaged Spinwriter. And no insurance. We didn't even have all of the serial numbers recorded. The police indicated we should have no hope of recovering anything.

I know that God, who made the heavens and the earth and all that is in them, is sovereign. I said, "Thank you for this, too. And thank you that we still have enough left to continue business. And that nothing irreplaceable was taken."

And we tried to to put the pieces back together. We bought insurance, and recorded all the remaining serial numbers. We made backup copies of critical software to be kept at other locations. We engraved our driver's license numbers on our equipment. We even installed an alarm system.

After about two hours with a screwdriver and needlenose pliers the Spinwriter was back in working condition. Almost as good as new...just one crippled foot where it landed when dropped. NEC makes durable gear. I spent another 8 hours figuring out how to talk to it with a serial interface card (with no documentation), and writing the driver program. Once it all worked, we were able to print the mailing labels for last month's AAL.

The burglary occured sometime after 8:30 pm, Wednesday night, May 25th. The next Wednesday night, after choir practice, we took some time to pray. Among other concerns, we prayed about the burglary. I suggested, "Let's pray that the burglars be caught and the things they took be recovered. It can't hurt to ask!" So we did.

The next day the police received a tip from an informer. They went to investigate, just in time to catch two 18-year-olds carrying computers from apartment to car. One of them was a well-known burglar, with at least six-year record. The equipment matched the description I had given them. Friday morning the investigator called: "We have some of your computers. You can come and pick them up at noon." Although a little dirty, none of the equipment or software was damaged. Two thirds of all that had been stolen was recovered! "A miracle", the police said. "Amen."

The following Monday the police called again. "We have some more." The third computer system, a brand new Apple //e with extended 80-column card, two disk drives, monitor, and Epson printer had been sold to a technically-minded friend (of the burglars) for only $100. Responding to the alternatives offered by our excellent police ("Return the computer, or go to jail"), the friend brought in all he had bought. Almost everything was back in our office!

Thursday, June 16th, I was called a third time. "We have another disk drive." They also had another FlipFile with about 15 more diskettes. Now all that is missing is a TI Programmer calculator and an old Panasonic Cassette Recorder.

Yes, God is sovereign, and also He cares about us as individuals. He allowed our things to be taken, but not everything. He gave us faith to ask for them to be returned. And He caused them to be returned. "Trust in the Lord with all your heart, and do not lean on your own understanding. In all your ways acknowledge Him, and He shall direct your paths." [Proverbs 3-5,6]

Eighty-Column SHOW Command.....Robert Bragner
Istanbul, Turkey

I make frequent use of the SHOW command for text files (see AAL July 1982), and I wanted to see it in 80-column glory on my shiny new //e. If you've tried it, you will have noticed that the command places a character in every other column on the 80-column screen, so you still only see 40-columns of data per line!

The reason is that the SHOW command code calls COUT1 at $FDF0 for its character output, and COUT1 knows nothing about 80-column output. By calling COUT ($FDED) instead, the text file output will be routed to whatever your current output device happens to be (including printer, 80-column display, etc.).

If you use the Applesoft on page 27 of that issue to load SHOW, all you need to do is change the ninth item on line 100 from 240 to 237.

Here is the modified POKEr, complete with the additions made by Bil Morgan in the June 83 issue, to save you hunting through all those back issues:

```
100 DATA 21,42319,32,163,162,169,141,32,237,253,32,142,
         174,240,5,32,140,166,208,243,76,252,162
110 DATA 23,44686,173,0,192,16,17,141,16,192,201,141,
         240,10,173,0,192,16,251,141,16,192,201,141,96
115 DATA 13,44709,224,0,240,4,162,2,208,2,162,4,76,3,171
116 DATA 3,43773,76,165,174
120 DATA 4,43140,83,72,79,215
130 DATA 2,43273,32,48
140 DATA 0
150 READ N : IF N THEN READ A : FOR I = 1 TO N
    : READ D : POKE A+I-1,D : NEXT : GO TO 150
```

Explanation of the new DOS Append Bug.....Tom Weishaar
Overland Park, Kansas

[Tom is author of ProntoDOS, published by Beagle Bros, an excellent speed-enhanced DOS which happens to be compatible with nearly everything. He also writes the monthly DOStalk column in Softalk Magazine.]

I was behind on my reading when I wrote, in the April Softalk DOStalk, about the changes Apple made to DOS 3.3 in the new //e release. At that time I noticed the routine used to calculate random access file position at $B331 had been modified, but the change looked insignificant to me.

It turns out this change was supposed to fix another bug in the Append command! The change was very well documented by Art Schumer in the August 1982 Call APPLE, page 57.

In pre-//e DOS, Append called on this random-access file position calculator to reset the position-in-file pointer. As you know, Append simply looks through a file byte-by-byte until it finds the end, which can be indicated either by a zero byte or by a lack of additional sectors.

When Append finds a zero byte in the file, it knows it has reached the end, but by then the position-in-file pointer is one byte beyond the zero and has to be backed up. Somebody once thought a call to the random-access file position calculator would be a good way to do this.

But on sequential files (the only kind you can append to) the File Manager uses a record length of one. Thus files longer than 32767 bytes come to this routine with more than 32767 "records", which is beyond what DOS normally allows. The calculation fails.

Schumer's patch gets it to calculate correctly right up to 65535. At that point it stops working for good. Apple tried to get around this in //e-DOS by throwing out Append's reliance on the random-access calculator. Instead they go back in and change the position-in-file pointer directly, then trick the File Manager into re-saving his workarea.

Problem: they only decrement the low byte of the position-in-file pointer. If the file-ending zero comes in the last byte of a sector, the high byte will have been advanced to point at the next sector. Since they don't decrement it, the position-in-file pointer is 256 bytes beyond where it should be. Uh Oh...!

I've been trying to get folks at Apple to recognize the problem, but Append doesn't appear to be one of their priorities. If they don't do something soon I'll publish a patch in Softalk. I'd do it now, but I fear treading where so many have failed before me.

The new 1983 edition of DOS 3.3.....Bob Sander-Cederlof

Co-incident with the release of the //e, Apple started shipping a slightly modified version of DOS 3.3. Three changes are evident: the sample programs have been moved to a separate diskette; a few instructions to kill 80-column display during a boot were added; and yet another patch to the APPEND command.

I booted an old DOS 3.3, and then used monitor move to make a copy in memory running from 5D00-7FFF of the DOS image. Then I booted the new DOS, which loaded into 9D00-BFFF. Using the monitor "V" command, I located all of the changes. It was a little tricky skipping over the variables and buffers, but with the aid of a well-worn copy of "Beneath Apple DOS" I managed. Here are all the changes I found:

```
 Old DOS 3.3                    New DOS 3.3
----------------------          ----------------------
A6BB:EA       NOP               A6BB:20 69 BA JSR $BA69
A6BC:EA       NOP
A6BD:EA       NOP
A6BE:A2 00    LDX #0            A6BE:unchanged
A6C0:8E C3 B5 STX $B5C3
A6C3:60       RTS
```

The code above is jumped to from one of the older APPEND patches at $B6A8. It used to be JMP $A6BC, and has been changed to JMP $A6BB to pick up the new JSR there.

The latter part of the file position calculator has been re-written to assure carry is clear before adding record size to previous position.

```
 Old DOS 3.3                    New DOS 3.3
----------------------          ----------------------
B33E:AD BF B5 LDA $B5BF         B33E:18       CLC
B341:8D EC B5 STA $B5EC         B33F:AD BF B5 LDA $B5BF
B344:6D E6 B5 ADC $B5E6         B342:8D EC B5 STA $B5EC
B347:8D E6 B5 STA $B5E6         B345:6D E6 B5 ADC $B5E6
B34A:AD C0 B5 LDA $B5C0         B348:8D E6 B5 STA $B5E6
B34D:8D ED B5 STA $B5ED         B34B:AD C0 B5 LDA $B5C0
B350:6D E4 B5 ADC $B5E4         B34E:8D ED B5 STA $B5ED
B353:8D E4 B5 STA $B5E4         B351:6D E4 B5 ADC $B5E4
B356:A9 00    LDA #0            B354:8D E4 B5 STA $B5E4
B358:6D E5 B5 ADC $B5E6         BE57:90 03    BCC $B35C
B35B:8D E5 B5 STA $B5E5         BE59:EE E5 B5 INC $B5E5
B35E:60       RTS               BE5C:60       RTS
                                BE5D:00 00    filler
```

Note that there was room for adding the CLC at the top, because of the in-efficiency of the original code.

QUICKTRACE

relocatable program traces and displays the actual machine operations, *while* it is running without interfering with those operations. Look at these *FEATURES:*

***Single-Step** mode displays the last instruction, next instruction, registers, flags, stack contents, and six user-definable memory locations.*

***Trace** mode gives a running display of the Single-Step information and can be made to stop upon encountering any of nine user-definable conditions.*

***Background** mode permits tracing with no display until it is desired. Debugged routines run at near normal speed until one of the stopping conditions is met, which causes the program to return to Single-Step.*

***QUICKTRACE** allows changes to the stack, registers, stopping conditions, addresses to be displayed, and output destinations for all this information. All this can be done in Single-Step mode while running.*

***Two optional display formats** can show a sequence of operations at once. Usually, the information is given in four lines at the bottom of the screen.*

***QUICKTRACE** is completely transparent to the program being traced. It will not interfere with the stack, program, or I/O.*

***QUICKTRACE** is relocatable to any free part of memory. Its output can be sent to any slot or to the screen.*

***QUICKTRACE** is completely compatible with programs using Applesoft and Integer BASICs, graphics, and DOS. (Time dependent DOS operations can be bypassed.) It will display the graphics on the screen while **QUICKTRACE** is alive.*

***QUICKTRACE** is a beautiful way to show the incredibly complex sequence of operations that a computer goes through in executing a program*

QUICKTRACE *$50*

Is a trademark of Anthro-Digital, Inc.

Written by John Rogers

See these programs at participating Computerland and other fine computer stores.

Anthro - Digital Software, Inc.
P.O. Box 1385 Pittsfield, MA 01202

Code executed at the end of a boot has been modified to clear 80-column mode in case you are booting in an Apple //e.

```
 Old DOS 3.3                 New DOS 3.3
---------------------       ---------------------
BFD6:4C 44 B7 JMP $B744     BFD6:20 76 BA JSR $BA76
                            BFD9:4C 44 B7 JMP $B744
```

Three patches were stuffed into the hole at $BA69.

```
          Called from $A6BB:

     BA69:AE 5F AA LDX $AA5F   If last command was
     BA6C:E0 1C    CPX #$1C       APPEND, clear flag
     BA6E:F0 05    BEQ $BA75      Not APPEND
     BA70:A2 00    LDX #0         Yes, APPEND
     BA72:8E 5D B6 STX $B65D      Clear APPEND flag
     BA75:60       RTS

          Called from $BFD6:

     BA76:A9 FF    LDA #$FF
     BA78:8D FB 04 STA $04FB      MODE = $FF
     BA7B:8D 0C C0 STA $C00C      80-column display OFF
     BA7E:8D 0E C0 STA $C00E      Alternate Char Set OFF
     BA81:4C 2F FB JMP $FB2F      Exit via monitor INIT

          Called from $B683:

     BA84:AD BD B5 LDA $B5BD      Previous file position
     BA87:8D E6 B5 STA $B5E6      LSB of fle position
     BA8A:8D EA B5 STA $B5EA      Record #
     BA8D:BA       TSX
     BA8E:8E 9B B3 STX $B39B      Save stack position
     BA91:4C 7F B3 JMP $B37F      Leave File Manager
```

Note that this last patch jumps into the file manager exit routine even though the file manager had not been entered. The purpose is to save a copy of the file manager workarea in the file buffer after patching the file position low-order byte. Seems to me that jumping directly to $AE7E, without the two lines saving the stack pointer above, would avoid the very dangerous step of jumping into the middle of a subroutine. But in any case, as Tom Weishaar points out, the code is wrong in that it does not recover the higher bytes of the file position. Will APPEND ever really be fixed?

A few months back I published a patch for faster LOAD, etc. in these pages which used the space from $BA69 through $BA95. I suggest you use the older version of DOS 3.3 for the time being. But eventually you may be forced to find another home for the fast LOAD patch.

D O W N L O A D I N G
CUSTOM CHARACTER SETS

One of the features 'hidden' in many printers available today is their ab ty to accept user-defined character sets. With the proper software, these **custom characters** are 'downloaded' from your Apple II computer to the printer in a fraction of a second. Once the printer has 'learned' these new characters, they will be remembered until the printer is turned off.

After the downloading operation, you can use your printer with virtually any word processor. Just think of the possibilities! There's nothing like having your own **CUSTOM CHARACTERS** to help convey the message. And you still have access to those built-in fonts as well! **Here's a quick look at some possible variations:**

	BUILT-IN	CUSTOM
10CPI:	AaBbCcDdEeFfGgHhIiJjKk	AaBbCcDdEeFfGgHhIiJjKk
12CPI:	AaBbCcDdEeFfGgHhIiJjKk	AaBbCcDdEeFfGgHhIiJjKk
17CPI:	AaBbCcDdEeFfGgHhIiJjKk	AaBbCcDdEeFfGgHhIiJjKk
5CPI:	AaBbCcDdEeFf	AaBbCcDdEeFf
6CPI:	AaBbCcDdEeFf	AaBbCcDdEeFf
8CPI:	AaBbCcDdEeFf	AaBbCcDdEeFf

And let's not forget Enhanced and Underined printing as well...

AaBbCcDdEeFfGgHhIiJjKk	AaBbCcDdEeFfGgHhIiJjKk
AaBbCcDdEeFfGgHhIiJjKk	AaBbCcDdEeFfGgHhIiJjKk

The Font Downloader & Character Editor software package has been developed by RAK-WARE to help you unleash the power of your printer. The basic package includes the downloading software with 4 fonts to get you going. Also included is a character editor so that you can turn your creativity loose. Use it to generate unique character fonts, patterns, symbols and graphics. A detailed user's guide is provided on the program diskette.

SYSTEM REQUIREMENTS:

* APPLE II, APPLE II Plus, APPLE //e or lookalike with 48K RAM
* 'DUMB' Parallel Printer Interface Board (like Apple's Parallel Printer Interface, TYMAC's PPC-100 or equivalent)

The Font Downloader & Editor package is only $39.95 and is currently available for either the Apple Dot Matrix Printer or C.Itoh 8510AP (specify printer). Epson FX-80 and OkiData versions coming soon. Enclose payment with order to avoid $3.00 handling & postage charge.

R A K - W A R E
41 Ralph Road West Orange New Jersey 07052

Say You Saw It In **APPLE ASSEMBLY LINE**!

S-C Macro Cross Assemblers

The high cost of dedicated microprocessor development systems has forced many technical people to look for alternate methods to develop programs for the various popular microprocessors. Combining the versatile Apple II with the S-C Macro Assembler provides a cost effective and powerful development system. Hobbyists and engineers alike will find the friendly combination the easiest and best way to extend their skills to other microprocessors.

The S-C Macro Cross Assemblers are all identical in operation to the S-C Macro Assembler; only the language assembled is different. They are sold as upgrade packages to the S-C Macro Assembler. The S-C Macro Assembler, complete with 100-page reference manual, costs $80; once you have it, you may add as many Cross Assemblers as you wish at a nominal price. The following S-C Macro Cross Assembler versions are now available, or soon will be:

Motorola:	6800/6801/6802	now	$32.50
	6805	now	$32.50
	6809	now	$32.50
	68000	now	$50.00
Intel:	8048	now	$32.50
	8051	now	$32.50
	8085	now	$32.50
Zilog:	Z-80	now	$32.50
RCA:	1802/1805	now	$32.50
Rockwell:	65C02	now	$20.00
DEC:	PDP-11/LSI-11	now	$50.00

The S-C Macro Assembler family is well known for its ease-of-use and powerful features. Thousands of users in over 30 countries and in every type of industry attest to its speed, dependablility, and user-friendliness. There are 20 assembler directives to provide powerful macros, conditional assembly, and flexible data generation. INCLUDE and TARGET FILE capabilities allow source programs to be as large as your disk space. The integrated, co-resident source program editor provides global search and replace, move, and edit. The EDIT command has 15 sub-commands combined with global selection.

Each S-C Assembler diskette contains two complete ready-to-run assemblers: one is for execution in the mother-board RAM; the other executes in a 16K RAM Card. The HELLO program offers menu selection to load the version you desire. The disks may be copied using any standard Apple disk copy program, and copies of the assembler may be BSAVEd on your working disks.

S-C Software Corporation has frequently been commended for outstandin support: competent telephone help, a monthly (by subscription) newsletter, continuing enhancements, and excellent upgrade policies.

S-C Software Corporation (214) 324-2050
P.O. Box 280300, Dallas, Texas, 75228

More opcodes for the S-C MACRO ASSEMBLER.....R.F. O'Brien

While using the assembler I felt that it was a pity that the BGE and BLT instructions had not been incorporated especially as it would only have meant an extra 6 bytes of code. This minimal extra overhead is because of the way opcodes are handled in the assembler.

Take for example the BRANCH opcode table, which resides in locations $EF5B-EF93. ($2E29-2E47 for RAM version.) [These addresses are for version 1.0]

This table is preceded by a 2-byte descriptor and ends as one would expect with a 00 as end-of-table marker. The descriptor in this case is 0302, i.e. 3-byte entries having a 2-byte name. The table holds the standard 8 6502-opcodes and the 10 Sweet-16 opcodes, interestingly the B of each instruction name has been dropped, saving 18 bytes.

The entries in the table consist of the last 2 letters of the instruction name followed by the hex code. In the case of 2-letter names the entry consists of the second letter plus a space ($20) followed by its hex code.

I decided that I could dispense with the SW-16 codes BM1 and BNM1 without suffering too much if I wanted to write Sweet-16 code in my programs. However, I found that to incorporate the new codes they would have to be placed between the 6502 codes and the SW-16 codes in the table.

It was just a matter of pushing the code for BR to BNZ up in RAM 6 bytes and slotting in the code for BLT and BGE.

To install the code for the two new opcodes just enter the following at any convenient location e.g.$4000 and BSAVE as BLT/BGE.CODE,A$4000,L$1F

```
:$4000:47 45 B0 4C 54 90      ("G E B0  L T 90")
:$:52 20 01 4E 43 02 43 20 03 50 20 04
:$:4D 20 05 5A 20 06 4E 5A 07 53 20 0C 00
```

To install in LC-Version just enter:

```
:$C083 C083         write enable card.
:BLOAD BLT/BGE.CODE,A$EF75
:$C080              write protect card.
```

To install in RAM-Version just enter:

```
:BLOAD BLT/BGE.CODE,A$2E29
```

If you never use Sweet-16 you only need to use the first 6 bytes of the above code. However, this will wipe out the BR and BNC codes in the table.

Now you can use either BCC or its synonym BLT (Branch if Less Than) and BCS or BGE (Branch if Greater than or Equal to) in your programs and have them assembled correctly without using macro definitions.

The load address for the patch file for version 1.1 will vary depending on which of the 8 versions you are patching:

	40-col	//e	Videx	STB-80
Motherboard	$31A9	318D	3274	329D
RAM Card	$F2C3	F2A7	F397	F3C0

Bobby Deen's Latest Stuff

Bobby Deen is a name you may remember seeing in these pages in several past issues. He will be entering Texas A & M University this fall. Bobby programmed most of the cross assembler modules for the S-C Macro Assembler, some parts of the S-C Word Processor, about half of the yet-to-be-released 18-digit commercial math package (S-C DP18), and parts of the CPR Training System we did for the American Heart Association. A man of many interests, Bobby also has produced some excellent music disks for the ALF music synthesizers (or any Alf compatibles, such as Applied Engineering), and now a fantastic "Othello" game.

His six-voice renditions of the William Tell Overture by Rossini and Tchaikovsky's Nutcracker Suite are outstanding, and the price is only $10. If you have a synthesizer, you ought to have Bobby's music.

Bobby's Othello program is available now for an introductory price of only $20. Of course he wrote it in assembly language, so it is FAST, and has excellent hi-res graphics. You select among six skill levels; Apple can suggest your next move; you can swap sides with the computer; you can modify the board in mid-game (cheat?); you can pit the computer against itself. Whenever two or more moves tie for the machine's best next move, Apple randomizes its choice. This way you never play the same identical game twice.

Order either of these disks from S-C Software.

Apple Assembly Line is published monthly by S-C SOFTWARE CORPORATION, P.O. Box 280300, Dallas, Texas 75228. Phone (214) 324-2050. Subscription rate is $15 per year in the USA, sent Bulk Mail; add $3 for First Class postage in USA, Canada, and Mexico; add $13 postage for other countries. Back issues are available for $1.50 each (other countries add $1 per back issue for postage).